A Translator for Structured Reports from DICOM Binary Format to XML Format

Jingkun Hu

K. P. Lee

Contact: jingkun.hu@philips.com

Healthcare Interoperability Project
Philips Research
345 Scarborough Road
Briarcliff Manor, New York, 10510-2099
USA

Key Words: **DICOM, Structured Reporting, XML, XSLT**

Abstract

Structured Reporting (SR), a supplement of the Digital Imaging and Communication in Medicine (DICOM) standard, specifies a standard form for medical reports. The format of these SR reports is DICOM and therefore they can be readily stored, transmitted, manipulated and viewed in the DICOM world. However, it is not convenient to access and view these reports in the non-DICOM world. The Extensible Markup Language (XML), a specification from the World Wide Web Consortium (W3C), specifies a universal format for structured information. XML provides an ideal solution for representing and viewing DICOM structured reports in the non-DICOM world, provided that an engine is available to translate these reports from DICOM binary format to XML format. Such an engine has been built on top of a DICOM library and extensible stylesheet language transformations (XSLT) components. It is implemented in Microsoft Visual C++ 6.0 on an NT platform. A set of tests has been run and results show that the engine works well.

The current prototype engine is a standalone application. However, it can easily be built into a server component in the DICOM world to connect to the non-DICOM world. It can also be used to translate DICOM messages other than structured reports.

Table of Contents

1. Introduction

The recently standardized DICOM Structured Reporting (SR) specification [1] is an initiative intended to support and add structure to conventional free text reports commonly used in medical diagnosis. The specification supports the transfer and storage of report documents and allows users to link text and other data to particular images and/or waveforms and to store the coordinates of findings so that users can see exactly what is being described in a report. Such reports are in DICOM binary format and can be stored, interchanged and viewed in any environment that supports the DICOM standard.

Users of SR can be radiologists, technicians, physicians, nurses, or patients who are geographically distributed and using different systems. Interoperability among different systems is an important issue. In general interoperability includes platform interoperability and data interoperability. This report focuses only on data interoperability and does not address platform interoperability issues handled with CORBA technology [7], or by other means.

In the medical community, support for DICOM is limited to certain specialties, primarily those dealing with images. In order to achieve data interoperability and effectively share information, a universal data format is needed. The Extensible Markup Language (XML) [4], standardized in February 1998, defines a universal data format and provides an ideal solution to sharing medical reports at the enterprise level, allowing medical reports to be accessed and viewed in the non-DICOM world.

We have implemented a translator, called DICOM2XML, which takes a structured report in DICOM binary format and produces an XML document. Software already exists [6] to do the reverse translation, taking a report in XML and reproducing it in DICOM binary format. Together these translators permit structured reports to be interchanged easily between DICOM and XML format.

This document describes the application scenario, design, and implementation of the DICOM2XML translator. The intended readers are software architects, developers, and project managers who are leveraging XML technology into healthcare information technology domain.

2. Application Scenario

Figure 1 shows an application scenario for DICOM2XML. The left half of the figure shows an environment with modalities and DICOM archives, for example, picture archiving and communication system (PACS), where structured reports are generated, stored, interchanged and viewed. The right half depicts an environment with little DICOM support but where there is still a desire to have access to the information in DICOM archives. DICOM2XML serves to convert medical reports from DICOM format to XML format. The resulting XML documents can be displayed on a variety of devices ranging from workstations to portable hand-held devices. For example, this component enables clinical physicians to easily access and view diagnosis reports that have been stored in DICOM repositories (e.g., PACS) from desktops, portable terminals, and other equipment that do not necessarily carry DICOM viewers and toolkits. This allows enterprise-wide access of the structured information.

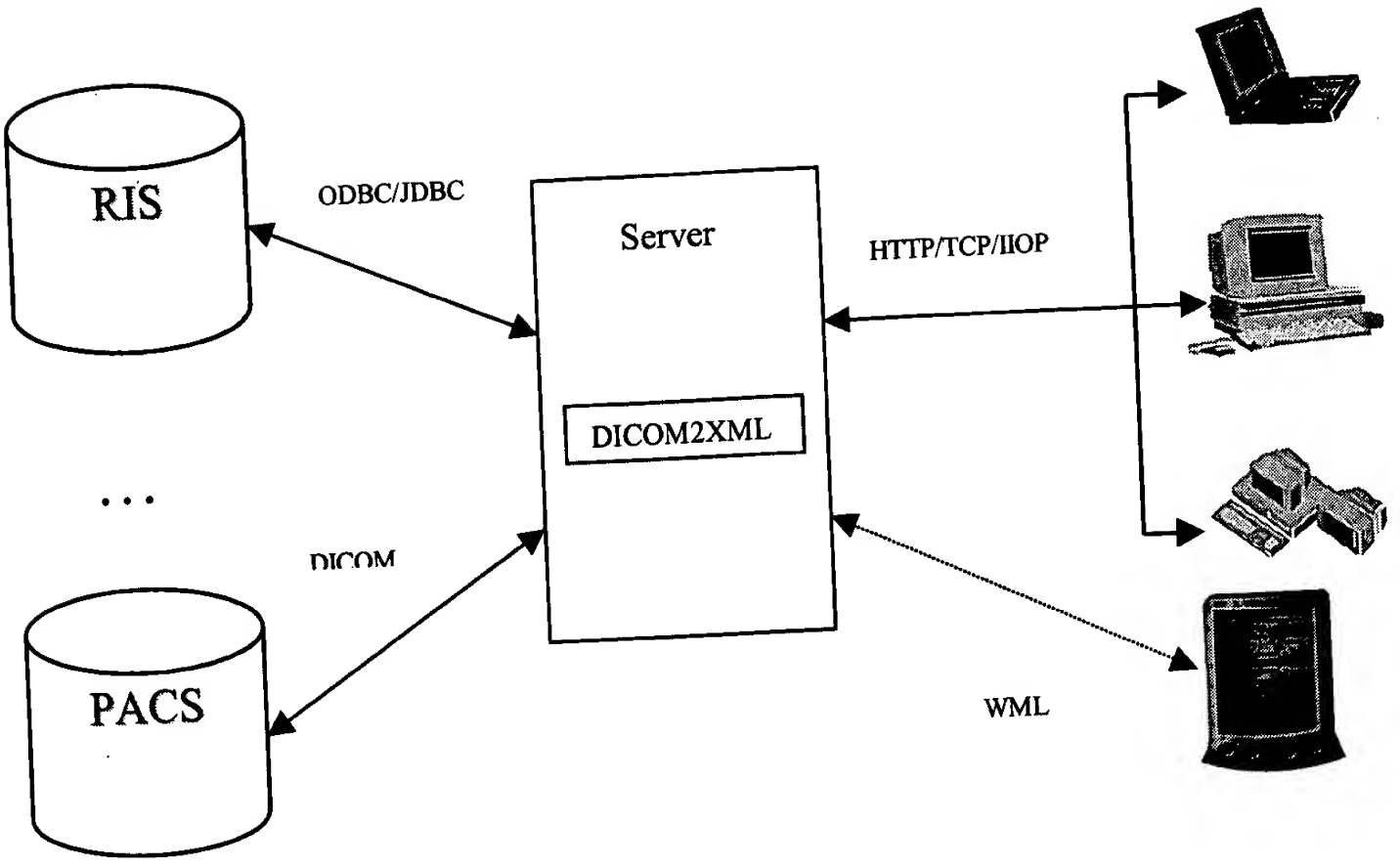

Figure 1. Application scenario of DICOM2XML translator

Although in this document we focus on DICOM SR, this engine can also be applied to other DICOM messages in the future, for example, appointment scheduling, patient information, or images. With the increasing development of telemedicine, WebMD, and in-home healthcare, an enterprise-wide distribution of medical information is needed. The DICOM format will be used more in the DICOM world where image acquisition plays an important role. But cooperation among physicians, clinicians, or consultants requires the exchange of patient data and medical reports where DICOM is not in common use. It will therefore be necessary to use a general open format for this information in the enterprise-wide world. This DICOM2XML translator provides such a function.

3. Design Decisions

There are a number of design scenarios in building this translator:

- One is to parse the input DICOM binary file and then build an XML document based on an XML document type definition (DTD) using an XML parser, for example a DOM parser [3] (Figure 2).
- Another is to parse the input DICOM binary file and generate a raw XML document based on the DICOM attribute information contained in the input, and then apply an XSL stylesheet using an XSLT engine [5] to build a target XML file (Figure 3).

In the first scenario, the DICOM parser takes a DICOM binary file, parses it, and outputs DICOM attribute information. Then the SR XML builder generates an XML DOM tree from scratch based on an SR XML DTD and the parsed DICOM attribute information by inserting DICOM attribute values into the proper places in the empty XML DOM tree.

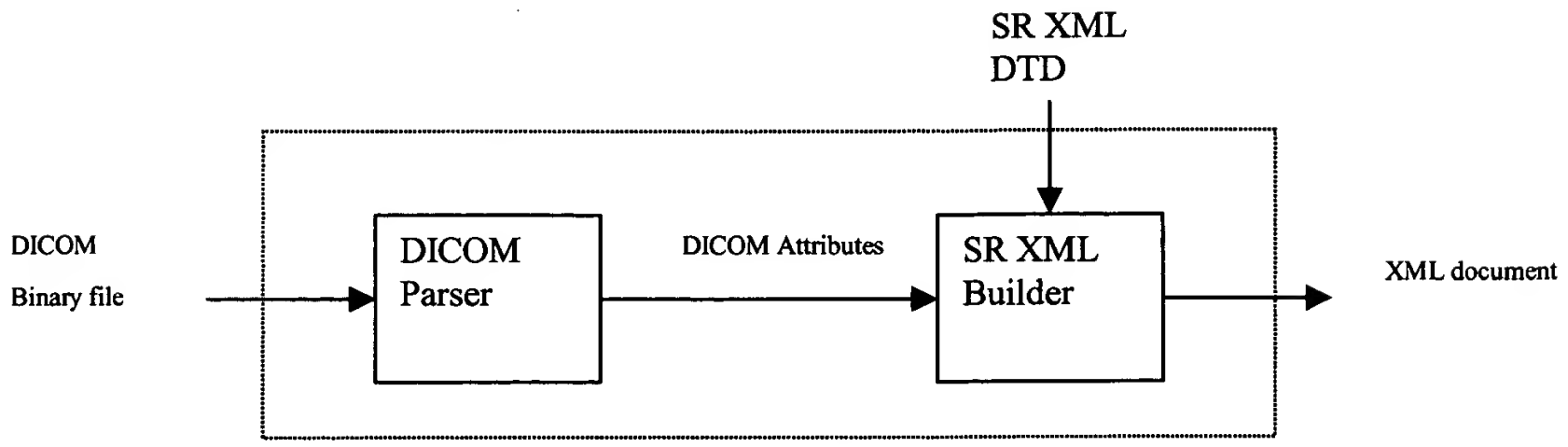

Figure 2. Software components and data flow diagram, scenario I

In the second scenario, the DICOM parser parses the DICOM binary file and the raw XML builder dumps DICOM information to a new XML document with simple structure. Then an XSLT engine, with a SR XSL stylesheet, transforms this raw XML document into a SR XML document.

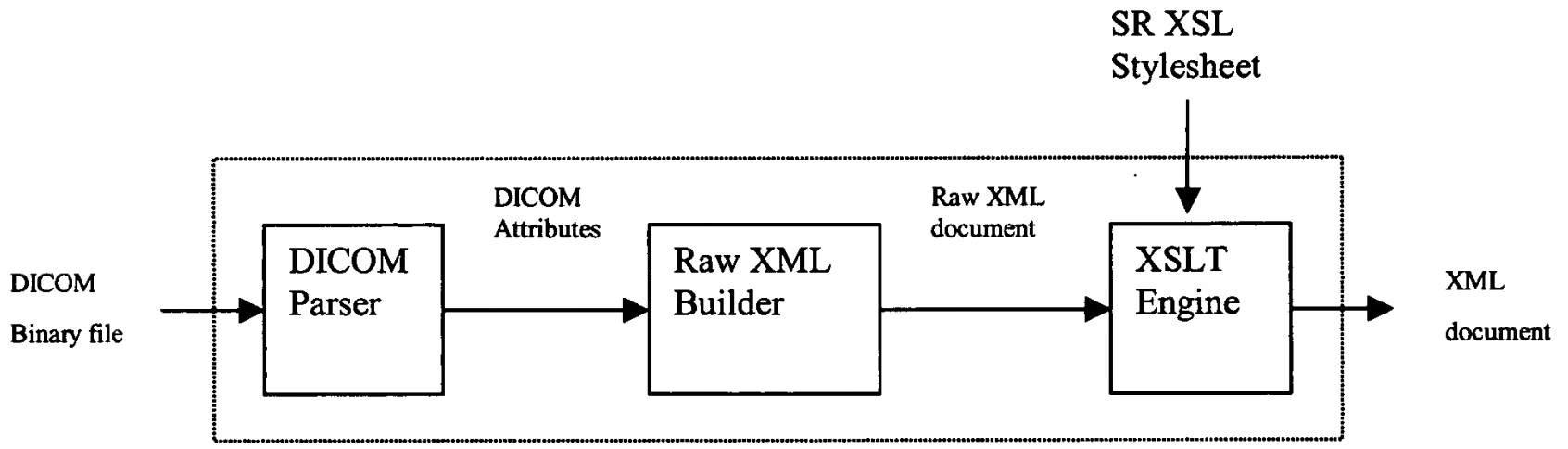

Figure 3. Software components and data flow diagram, scenario II

A DICOM parser is needed in both cases. The first case requires only two steps, but the SR XML builder will need to incorporate coding logic dependent on the DTD used, thus reducing the reusability of this component. In the second case, the raw XML builder is simple to implement by adding tags to the parsed DICOM attributes. An XSLT engine is applied with a SR XSL stylesheet to produce the final SR XML document. There are numerous XSLT engines available, either commercial or free. In this case, most application-specific logic resides in stylesheets and therefore maintenance is simplified. For this reason we selected the second approach.

4. Component Selection

This engine is developed under a Microsoft Visual C++ 6.0 environment, which limits the component selection to C or C++ versions of available tools.

The MergeCOM3 toolkit (version 3.0) [2] is a widely used commercial product for DICOM implemented in ANSI C from Merge Corp. Philips Medical Systems has purchased a corporate license. For this reason we choose the MergeCOM3 toolkit for this implementation.

There are a number of XML parsers currently available, many of which are open source and can be freely used. In order to simplify engine implementation and comply with XML standards, a C++ XML parser, Xerces version 1.1, from Apache Software Foundation [9], is selected.

5. Component Diagram

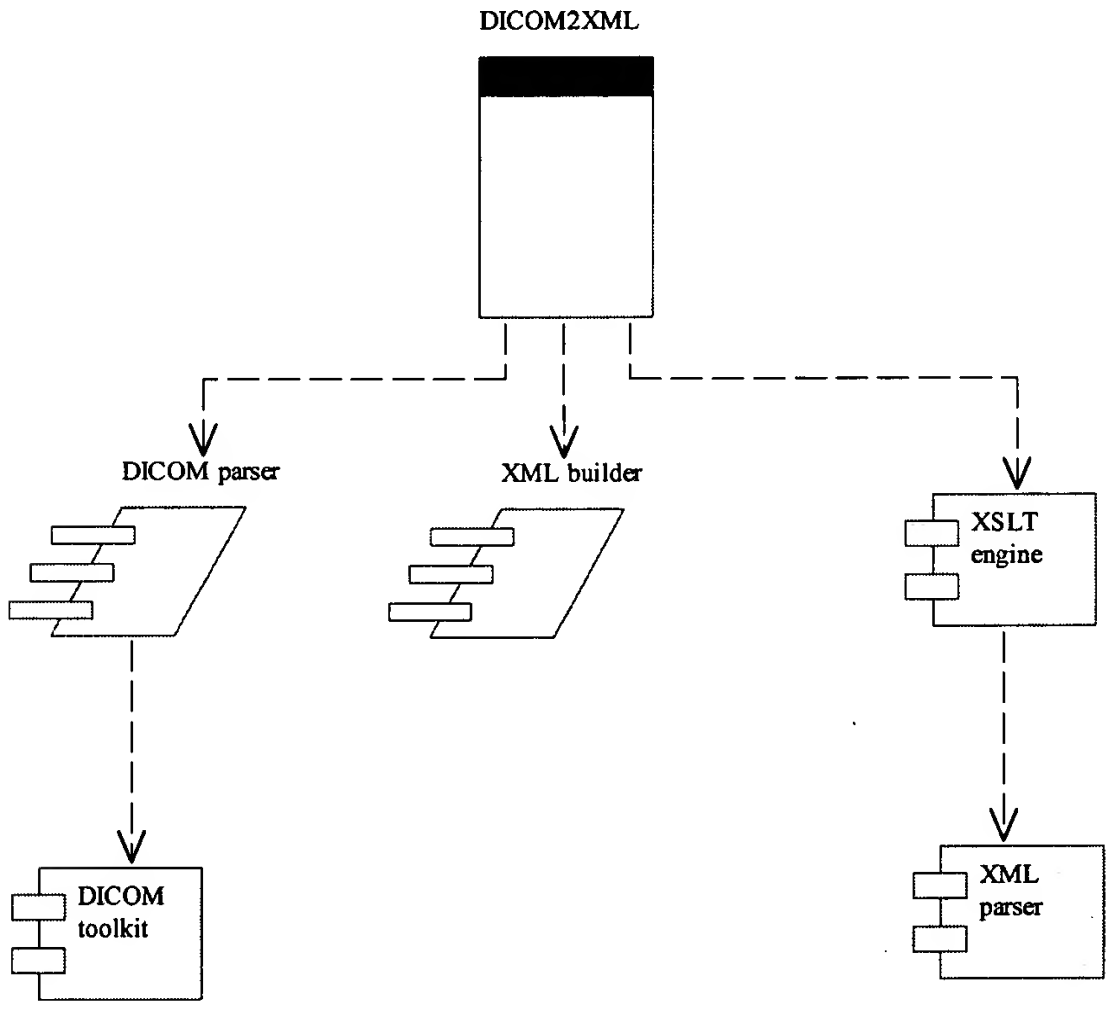

Figure 4. Component diagram of DICOM2XML

Figure 4 shows the component diagram of the SR DICOM2XML translator. DICOM2XML, the main component, processes the command line and invokes the different components sequentially. The DICOM parser loads a DICOM binary file and parses it using the Merge DICOM toolkit. The XML builder dumps the result from the DICOM parser into a raw XML document following a set of rules. And then the XSLT engine takes the raw XML document and a SR XSL stylesheet to generate a SR XML document.

The module design of the DICOM parser is as follows:

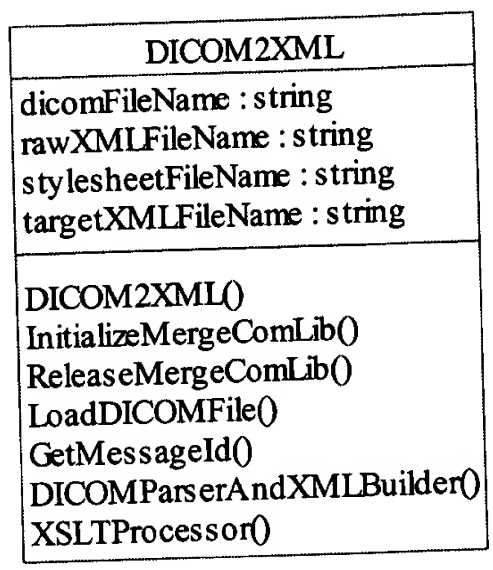

Figure 5. Class diagram of DICOM2XML module

Figure 5 shows the complete member operations and variables of the DICOM2XML module. The detailed descriptions are as follows:

dicomFileName
A string which is the path name of a SR DICOM binary file.

targetXMLFileName
A string which is the path name of an SR XML document.

rawXMLFileName
A string which is the path name of a raw XML document.

stylesheetFileName
A string which is the path name of an SR XSL stylesheet.

DICOM2XML()
Constructor of the class DICOM2XML. It takes three parameters, SR DICOM binary input, SR XSL stylesheet, and targets SR XML document.

InitializeMergeComLib()
Initializes the MergeCOM toolkit library. It returns true if it is successful. Otherwise, it returns false.

ReleaseMergeComLib()
Frees all the memory used by MergeCom toolkit. It returns true if it is successful. Otherwise, it returns false.

LoadDICOMFile()
Loads the SR DCIOM binary file and converts it into a DICOM message object.

DICOMParserAndXMLBuilder()
Recursively parses a DICOM message object and writes the parsed DICOM attributes to an XML document.

GetMessageId()

Returns the DICOM message object identification.

XSLTProcessor()

Transforms the raw XML document into an SR XML document by applying an SR XSL stylesheet.

6. Implementation

The software for the DICOM2XML engine includes one header file and two implementation files:

`DICOM2XML.h` – header file defining all the member functions of the translator.

`DICOM2XML.cpp` – implementation of `DICOM2XML.h`.

`DICOM2XMLMain.cpp` – main program.

Compiling `DICOM2XML.h` and `DICOM2XML.cpp` produces a static library `XML2DICOM.lib`. The main program calls the functions in this library to do the translation.

The rules by which a raw XML file is generated are as follows:

1. The root element of the raw XML document is named 'report'.
2. Each DICOM attribute is written out as an XML element with four attributes for a non-Sequence attribute: CodingScheme, CodeId, Value, and ValueType, and three attributes for a Sequence attribute: CodingScheme, CodeId, and ValueType.
3. A DICOM Item or Sequece Item contains other DICOM attributes. Rules 2 and 3 are applied recursively.
4. The element name is generated as follows: (a) Change all the upper case letters in the DICOM attribute name to lower case; (b) Replace the blank space between two words in the name with an underscore; (c) Remove apostrophes and brackets; (d) Replace hyphen (-) and slash (/) with underscore (_).
5. The value of CodingScheme is fixed as "DCMTAG". The value of CodeId and value of Value are directly taken from the DICOM parser.
6. The value of ValueType is determined from the mapping table below:

DICOM Data Type	SR XML ValueType
AE, AS, CS, DA, DS, DT, IS, LO, LT, PN, SH, ST, TM, UT, UI, UN, OB, OW, OL	string
SS,US	unsigned short
FL, FD	float
AT, SL, UL	unsigned long
SQ	sequence

These rules are compliant with the SR XML DTD described in [8].

7. SR XSL Stylesheet

A set of XSL stylesheets is created for each service-command pair. SR has three information object definition (IOD) modules: basic text SR, enhanced SR, and comprehensive SR. They all contain the same information entities (IEs) and modules as shown in the following table:

IE	Module
Patient	Patient
	Specimen Identification
Study	General Study
	Patient Study
Series	SR Document Series
Equipment	General Equipment
Document	SR Document General
	SR Document Content
	SOP Common

The SR XSL stylesheets are created based on the structure of the raw XML documents and SR XML DTD. They are modularized with a stylesheet for each of the IEs to enable reuse of the common stylesheets in the three IOD modules.

The high level structure of the SR XSL stylesheets is as follows:

```
<xsl:include href="Patient_IE.xsl"/>
<xsl:include href="Study_IE.xsl"/>
<xsl:include href="Series_IE.xsl"/>
<xsl:include href="Equipment_IE.xsl"/>
<xsl:include href="Document_IE.xsl"/>
<!-- template for top-level element -->
<xsl:template match="report">
      <SRDocument>
            <xsl:attribute name="report_id"><xsl:value-of select="@id"/>
            <xsl:attribute name="report_date"><xsl:value-of select="@date"/>
            <xsl:call-template name="Patient_IE"/>
            <xsl:call-template name="Study_IE"/>
            <xsl:call-template name="Series_IE"/>
            <xsl:call-template name="Equipment_IE"/>
            <xsl:call-template name="Document_IE"/>
      </SRDocument>
</xsl:template>
```

The Patient_IE, Study_IE, Series_IE, and Equipment_IE stylesheets can be reused by all three SR XSL stylesheets. The following shows the core part of `Patient_IE` template, defined in `Patient_IE.xsl`:

```
<xsl:template name="Patient_IE">
      <Patients>
            <Patient>
             <xsl:call-template name="patients_name_template"/>
             <xsl:call-template name="patient_id_template"/>
             <xsl:call-template name="patients_birth_date_template"/>
             <xsl:call-template name="patients_sex_template"/>
             <xsl:apply-templates select="patients_birth_time"/>
             <xsl:apply-templates select="other_patient_ids"/>
             <xsl:apply-templates select="other_patient_names"/>
             <xsl:apply-templates select="ethnic_group"/>
             <xsl:apply-templates select="patient_comments"/>
            </Patient>
       </Patients>
</xsl:template>
```

The Document_IE stylesheet is more complicated than the other and is decomposed into three simpler stylesheets: `SR_Document_General_Module.xsl`, `SR_Document_Content_Module.xsl`, and `SOP_Common.xsl`:

```
<xsl:include href="SR_Document_General_Module.xsl"/>
<xsl:include href="SR_Document_Content_Module.xsl"/>
<xsl:include href="SOP_Common_ Module.xsl"/>
<xsl:template name="Document_IE">
      <Document>
       <SRDocumentGeneral>
            <xsl:call-template name="SR_Document_General_Module"/>
       </SRDocumentGeneral>
       <SRDocumentContent>
            <xsl:call-template name="SR_Document_Content_Module"/>
       </SRDocumentContent>
       <SOPCommon>
            <xsl:call-template name="SOP_Common_Module"/>
       </SOPCommon>
      </Document>
</xsl:template>
```

The difference among the different SR IODs is in the SR Document Content Module. Each SR IOD has a separate `SR_Document_Content_Module.xsl`.

8. Test Results

Several test files generated from the SR XML2DICOM engine [6] are used, so both the original SR XML documents and re-generated SR XML documents are available. One of them is based on the example in the SR specification [1]. The others are from the numerical reports of ultrasound obstetrics. The test results demonstrated that the output from the DICOM2XML engine completely matches the original input to the SR XML2DICOM engine.

9. Conclusions

Currently most DICOM reports/archives are in DICOM format, which limits their enterprise-wide distribution. An engine converting medical reports from DICOM format to XML format increases distribution of these reports/archives to environments with minimal DICOM support. Thus medical reports can be distributed to any healthcare site equipped with a Web browser. Further, SR allows reports

to be processed in a structured way allowing anticipated uses ranging from outcomes analysis to epidemiological research.

The design scenario we chose uses COTS XSLT components, it is more flexible and can easily adapt to updated versions of XSLT engines. In the current implementation, the Xalan XSLT engine from Apache is used, but it can easily be replaced by others.

The rules to dump DICOM binary files into raw XML documents are the same rules as in mapping the SR UML model to XML representation [8], which makes the creation of the SR XSL stylesheets more convenient. The resulting modularized XSL stylesheets are easy to maintain, and can be reused other DICOM services.

The test results showed that this DICOM2XML translator works very well. The next step is to build a software component based on this prototype and integrate it into a DICOM data server linked to DICOM archives.

10. References

[1] Digital Imaging and Communications in Medicine (DICOM), Supplement 23: Structured Reporting Storage SOP Classes, Final Text, DICOM Standards Committee, Rosslyn, VA, April, 2000.

[2] DICOM toolkit documentation, Merge Technology Inc, http://www.merge.com.

[3] Document Object Model (DOM), http://www.w3c.org/DOM.

[4] Extensible Markup Language (XML) 1.0 W3C Recommendation 10-February-1998, http://www.w3c.org.

[5] Extensible Stylesheet Language (XSL), http://www.w3c.org/Style/XSL.

[6] J. Hu, Prototyping of SR XML to DICOM engine, TN-2000-050, Philips Research USA, Nov 2000.

[7] Object Management Group, http://www.omg.org.

[8] A. Tirado-Ramos & J. Hu, UML Modeling and XML DTD representation of DICOM Structured Reporting, TR-2000-019, Philips Research USA, Aug 2000.

[9] Xerces XML Parser, Apache Software Foundation, http://xml.apache.org.

11. Glossary

CORBA	Common Object Request Broker Architecture
DICOM	Digital Imaging and Communications in Medicine
DTD	Document Type Definition
IE	Information Entity
IOD	Information Object Definition
PACS	Picture Archiving and Communication System
SR	Structured Reporting
XML	Extensible Markup Language
XSL	Extensible Stylesheet Language
XSLT	XSL Transformations